图文 39 基于MQ实现订单系统的第三方系统异步对接改造，解

528 人次阅读 2019-11-27 07:00:00

详情 评论

基于MQ实现订单系统的第三方系统异步对接改造，解耦架构完成！

石杉老哥重磅力作：《互联网java工程师面试突击》（第3季）**【强烈推荐】**：

全程真题驱动，精研Java面试中**6大专题的高频考点**，从面试官的角度剖析面试

（点击下方蓝字试听）

《互联网Java工程师面试突击》（第3季）

1、下一步要解决订单系统哪个问题？

小猛在推动完成了订单系统的核心流程的异步化改造后，让核心流程的性能一下子提升了10倍以上，从原来需要1秒甚至几秒才能完成，到现在核心流程只需要执行订单状态更新以及库存扣减两个步骤，以及发送一个消息到RocketMQ里去，仅仅需要耗时100ms。

这个优化的效果让所有人都很满意，因此明哥继续支持小猛对下一个问题引入MQ进行改造

那么接着应该解决订单系统哪个问题呢？小猛这个时候回顾了一下之前明哥列出来的问题列表：

~~下单核心流程环节太多，性能较差~~
订单退款的流程可能面临退款失败的风险
关闭过期订单的时候，存在扫描大量订单数据的问题
跟第三方物流系统耦合在一起，性能存在抖动的风险
大数据团队要获取订单数据，存在不规范直接查询订单数据库的问题
做秒杀活动时订单数据库压力过大

此时小猛已经在上面的列表中划掉了第一项，还剩下5个问题。

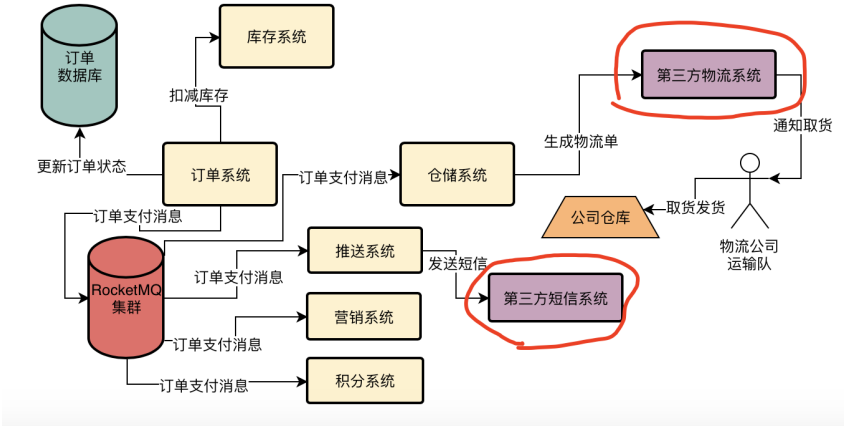
这个时候他将目光移动到了第四条，也就是“跟第三方系统耦合在一起，性能存在抖动的风险”。

这个时候他仔细思考了一下，将目光转移到了目前的订单核心流程图上去。

他发现在这个订单核心流程图里，可以非常清晰的看到，订单系统间接耦合的第三方系统有两个：

一个是第三方短信系统，是用来推送短信给用户的，一个是第三方物流系统，用生成物流单通知物流公司来收货和配送的。

我们看下面的图画圈的地方，就标识出来了这两个东西。



实际上如果按照以前最早的订单核心流程，订单系统会同步调用推送系统，然后推送系统调用第三方短信系统去发送短信给用户，接着订单系统会同步调用仓储系统，然后仓储系统调用第三方物流系统去生成物流单以及通知发货。

因此在最早的流程中，其实订单系统是间接性的跟第三方短信系统和第三方物流系统耦合在一起的，这样的话，一旦第三方系统出现了性能抖动就会影响到订单系统的性能。

比如正常第三方短信系统发送一个短信，只需要100ms，结果某一天突然性能下降变成发送短信需要1s了，此时会连带导致订单系统的性能也急剧下降。

但是似乎小猛仔细凝视上面的架构图，他发现现在引入了MQ之后，似乎订单核心流程已经变化了，然后对两个第三方系统的耦合问题似乎也有所改变了

既然如此，不如下一个问题就看看这个第三方系统的耦合问题如何解决吧！

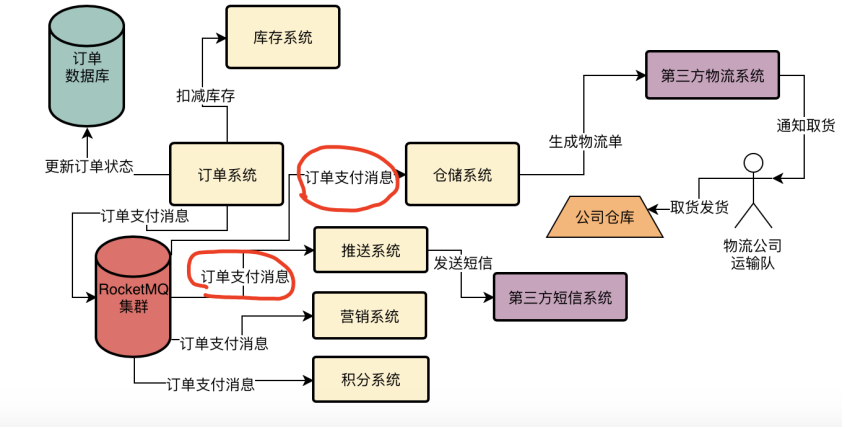
2、现在订单系统还跟第三方系统耦合吗？

小猛盯着上面的架构图，开始思考一个问题，现在订单系统还跟两个第三方系统耦合在一起吗？

因为实际上订单系统现在已经不需要直接调用推送系统和仓储系统了，仅仅只是发送一个消息到RocketMQ而已

所以小猛突然灵光一闪发现了一个问题，那就是订单系统跟第三方系统耦合导致性能抖动的问题，其实已经解决了！

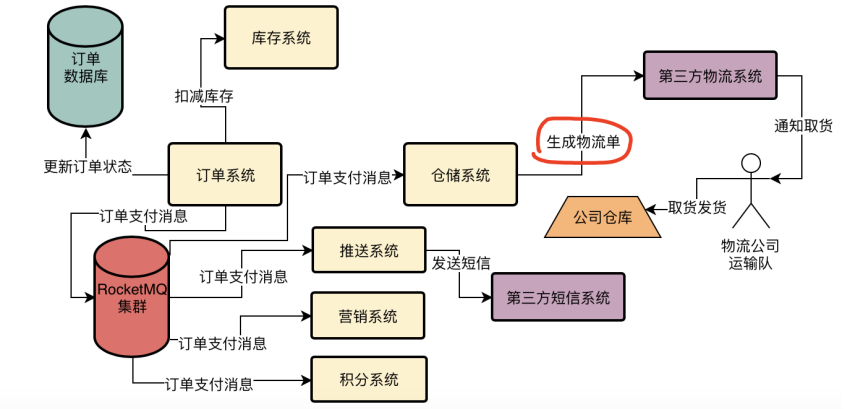
因为通过引入MQ到架构里，现在订单系统已经成功的跟推送系统以及仓储系统解耦了，如下面图里的红圈所示。



所以现在订单系统已经跟仓储系统和推送系统没关系了，只不过是仓储系统自己跟第三方物流系统耦合，推送系统自己跟第三方短信系统耦合而已！

此时即使第三方系统出现了严重的性能抖动，甚至是接口故障无法访问，也跟订单系统没任何关系！

因此比如第三方物流系统出现了性能抖动，此时只会影响到仓储系统而已，仓储系统调用第三方物流系统的接口时会出现短暂性的速度较慢的问题，如下图中红圈所示。



所以实际上在订单系统问题列表中的与第三方系统耦合的问题，已经被解决掉了，小猛想到这里，又把下面的第4项给划掉了。

~~下单核心流程环节太多，性能较差~~
订单退款的流程可能面临退款失败的风险
关闭过期订单的时候，存在扫描大量订单数据的问题
~~跟第三方系统耦合在一起，性能存在抖动的风险~~
大数据团队要获取订单数据，存在不规范直接查询订单数据库的问题
做秒杀活动时订单数据库压力过大

3、对RocketMQ的使用做一点进一步的探索？

如此轻松就解决了一个订单系统的耦合问题，小猛觉得真是太愉快了，但是另外一方面小猛觉得既然这个任务如此轻松就完成了，那么不如抽时间对RocketMQ的使用做一点进一步的探索，然后给各个团队的兄弟做一次技术分享

这样可以让其他团队的兄弟都了解一下RocketMQ的使用方式都有哪几种，相信对各个团队初步使用RocketMQ是比较有意义的。

说到做到，小猛很快就对RocketMQ的使用方式做了进一步的探索，并且梳理出来了一份技术分享PPT，然后约了各个部门的同事，做了一次技术分享。

4、什么叫做同步发送消息到RocketMQ？

今天分享的第一个点：什么叫做同步发送消息到RocketMQ？

小猛打开了一个代码片段如下所示：

```
public class RocketMQProducer {

       // 这个是RocketMQ的生产者类，用这个就可以发送消息到RocketMQ
       private static DefaultMQProducer producer;

       static {
              // 这里就是构建一个Producer实例对象
              producer = new DefaultMQProducer("order_producer_group");
              // 这个是为Producer设置NameServer的地址，让他可以拉取路由信息
              // 这样才知道每个Topic的数据分散在哪些Broker机器上
              // 然后才可以把消息发送到Broker上去
              producer.setNamesrvAddr("localhost:9876");
              // 这里是启动一个Producer
              producer.start();
       }

       public static void send(String topic, String message) throws Exception {
       // 这里是构建一条消息对象
           Message msg = new Message(
               topic, // 这就是指定发送消息到哪个Topic上去
             "",// 这是消息的Tag，我们后续再讲
             message.getBytes(RemotingHelper.DEFAULT_CHARSET) // 这是消息
           );
            // 利用Producer发送消息
           SendResult sendResult = producer.send(msg);
           System.out.printf("%s%n", sendResult);
       }
     }

}
```

大家可以看到上面的代码片段就是我们目前发送消息到RocketMQ里去的代码，实际上这种方式就是所谓的同步发送消息到MQ

那么什么叫同步发送消息到MQ里去?

所谓同步，意思就是你通过这行代码发送消息到MQ去，SendResult sendResult = producer.send(msg)，然后你会卡在这里，代码不能往下走了

你要一直等待MQ返回一个结果给你，你拿到了SendResult之后，接着你的代码才会继续往下走。

这个就是所谓的同步发送模式。

5、什么叫做异步发送消息到RocketMQ?

接着我们看一下所谓的异步发送消息到MQ的代码是什么样的

首先在构造Producer的时候加入下面红框中的代码：

```
// 这里就是构建一个Producer实例对象
producer = new DefaultMQProducer("order_producer_group");

// 这个是为Producer设置NameServer的地址，让他可以拉取路由信息
// 这样才知道每个Topic的数据分散在哪些Broker机器上
// 然后才可以把消息发送到Broker上去
producer.setNamesrvAddr("localhost:9876");

// 这里是启动一个Producer
producer.start();

// 设置异步发送失败的时候重试次数为0
producer.setRetryTimesWhenSendAsyncFailed(0);
```

接着把发送消息的代码改成如下所示：

```
producer.send(message, new SendCallback() {
    @Override
    public void onSuccess(SendResult sendResult) {

    }
    @Override
    public void onException(Throwable e) {

    }
});
```

这个意思就是说，你把消息发送出去，然后上面的代码就直接往下走了，不会卡在这里等待MQ返回结果给你！

然后当MQ返回结果给你的时候，Producer会回调你的SendCallback里的函数，如果发送成功了就回调onSuccess函数，如果发送失败了就回调onExceptino函数。

这个就是所谓的异步发送，异步的意思就是你发送消息的时候不会卡在上面那行代码等待MQ返回结果给你，会继续执行下面的别的代码，当MQ返回结果给你的时候，会回调你的函数！

6、什么叫做发送单向消息到RocketMQ?

还有一种发送消息的方法，叫做发送单向消息，就是用下面的代码来发送消息：

```
producer.sendOneway(msg);
```

这个sendOneway的意思，就是你发送一个消息给MQ，然后代码就往下走了，根本不会关注MQ有没有返回结果给你，你也不需要MQ返回的结果，无论发送的消息是成功还是失败，都不关你的事。

7、这几种发送消息的方式到底该用哪一种?

此时肯定会有很多人有疑问了，你告诉了我三种消息发送的模式，那么这个时候到底应该要用哪一种呢?

大家别着急针对这个问题，后续我们要结合消息不丢失、消息顺序性等案例场景来分析，你到底是适合同步消息？异步消息？还是单向消息？这个问题需要在后续的讲解中逐步展开。

目前大家只要知道，发送消息给MQ有这几种方式就可以了。

8、什么叫做Push消费模式?

小猛说完了发送消息给MQ的几种模式，接着讲到了消费消息的问题，首先他先打开了下面的代码片段，也就是目前各个系统从RocketMQ中消费消息的代码片段：

```
// 这是RocketMQ消费者实例对象
// “credit_group”之类的就是消费者分组
// 一般来说比如积分系统就用“credis_consumer_group”
// 比如营销系统就用“marketing_consumer_group”
// 以此类推，不同的系统给自己取不同的消费组名字
DefaultMQPushConsumer consumer = new   DefaultMQPushConsumer("credit_group");

// 这是给消费者设置NameServer的地址
// 这样就可以拉取到路由信息，知道Topic的数据在哪些broker上
// 然后可以从对应的broker上拉取数据
consumer.setNamesrvAddr("localhost:9876");

// 选择订阅“TopicOrderPaySuccess”的消息
// 这样会从这个Topic的broker机器上拉取订单消息过来
consumer.subscribe("TopicOrderPaySuccess");

// 注册消息监听器来处理拉取到的订单消息
// 如果consumer拉取到了订单消息，就会回调这个方法教给你处理
consumer.registerMessageListener(new MessageListenerConcurrently() {

    @Override
    public ConsumeConcurrentlyStatus consumeMessage(List<MessageExt> msgs, ConsumeConcurrentlyContext context) {

        // 在这里对获取到的msgs订单消息进行处理
        // 比如增加积分、发送优惠券、通知发货，等等
        return ConsumeConcurrentlyStatus.CONSUME_SUCCESS;
    }
});

// 启动消费者实例
consumer.start();
System.out.printf("Consumer Started.%n");
```

大家注意里面Consumer的类名：DefaultMQPushConsumer。

从类名中我们可以提取出来一个关键的信息：Push。其实从这里我们就能看出来，当前我们使用的消息消费实际上是Push模式。

那么什么是Push消费模式呢?

其实很简单，就是Broker会主动把消息发送给你的消费者，你的消费者是被动的接收Broker推送给过来的消息，然后进行处理。

这个就是所谓的Push模式，意思就是Broker主动推送消息给消费者。

9、什么叫做Pull消费模式?

小猛接着打开了下面的代码片段，这是RocketMQ官方提供的Pull消费模式的代码片段：

```
DefaultMQPullConsumer consumer = new DefaultMQPullConsumer("test_consumer_group");
consumer.start();

Set<MessageQueue> mqs = consumer.fetchSubscribeMessageQueues("TopicTest1");
for (MessageQueue mq : mqs) {
    System.out.printf("Consume from the queue: %s%n", mq);
    SINGLE_MQ:
    while (true) {
        try {
            PullResult pullResult = consumer.pullBlockIfNotFound(
                mq, null, getMessageQueueOffset(mq), 32);
            System.out.printf("%s%n", pullResult);
            putMessageQueueOffset(mq, pullResult.getNextBeginOffset());
            switch (pullResult.getPullStatus()) {
                case FOUND:
                    break;
                case NO_MATCHED_MSG:
                    break;
                case NO_NEW_MSG:
                    break SINGLE_MQ;
                case OFFSET_ILLEGAL:
                    break;
                default:
                    break;
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }}
```

在上述代码中，我们可以看到使用的Consumer类是DefaultMQPullConsumer，从名字里就可以看到使用了Pull消费模式。

也就是说，Broker不会主动推送消息给Consumer，而是消费者主动发送请求到Broker去拉取消息过来。

10、到底该使用Push模式还是Pull模式来消费?

看到这里又有很多朋友要问了，那么我们到底应该使用Push模式还是Pull模式来消费?

其实这个要看具体的场景了，我们现在暂时先不要做太多的深究，只要先知道有两种消费模式就可以了。

后面我们会通过更多的案例分析来思考在什么场景下应该使用哪种消费模式，所以大家不用太心急。

11、一点小小的总结

到目前为止，我们发现通过引入MQ到订单核心流程中，已经解决了两个问题：

核心流程环节过多性能较差的问题
跟第三方系统耦合导致性能容易抖动的问题

另外我们还探索了RocketMQ使用的几种方式，包括同步发送消息、异步发送消息、单向发送消息，Push消费模式以及Pull消费模式。

End

狸猫技术窝其他精品专栏推荐：

《从零开始带你成为JVM实战高手》

《21天Java 面试突击训练营》（分布式篇）（现更名为：**互联网Java工程师面试突击第2季**）

互联网Java工程师面试突击（第1季）

重要说明：

如何提问：每篇文章都有评论区，大家可以尽情在评论区留言提问，我会逐一答疑

如何加群：购买了狸猫技术窝专栏的小伙伴都可以加入**狸猫技术交流群**

具体加群方式，请参见目录菜单下的文档：《付费用户如何加群？》**（购买后可见）**